Computer Science

COMPSCI 372 S2 C – Exercise Sheet 2
Sample Solution

Q1: In this exercise you will write a program to read a number of vectors from a file and to sort them by length.

(a) Create a new .NET solution 'Exercise1Q1.sol' which consists of one file 'Exercise1Q1.cpp'. In the file define a new type Vector2D which is a structure with two variables x and y of type double.

```
typedef struct{
    double x;
    double y;
} Vector2D;
```

(b) Add a function `length()` which computes the length of its argument of type `Vector2D`.

```
double length(Vector2D v){ return sqrt(v.x*v.x+v.y*v.y);}
```

(c) Define yourself a file `MyVectors.txt` which contains an integer *n* specifying the number of vectors in that file and then 2**n* doubles representing the vectors. Define two global variables

```
int numVectors;
Vector2d *vectors;
```

and write a method which reads in the size of the array from the input file, allocates memory for `vectors`, and then reads in the values of the vectors from the input file.

Hint: Put the file `MyVectors.txt` is in the same directory as your application. It is a good idea to read the input with `fscanf()` since that way you don't have to worry about spaces, tabs, and end-of-line characters.

```
void readArray(char *fileName){
    FILE* inputStream=fopen(fileName,"r");
    if (!inputStream) fprintf(stderr,"Could not open file %s", fileName);
    else {
        fscanf(inputStream,"%d",&numVectors);
        vectors=(Vector2D*) malloc(numVectors*sizeof(Vector2D));
        for(int i=0;i<numVectors;i++)
            fscanf(inputStream,"%lf %lf", &vectors[i].x, &vectors[i].y);
    }
}
```

(d) Write a function

```
int compareLength(const void* v1, const void* v2)
```

so that it returns:

- an integer less than 0, if the length of v1 is less than the length of v2
- an integer greater than 0, if the length of t1 is greater than the length of v2
- an integer equal to 0, if the length of t1 is equal to the length of v2

Hint: The arguments of `compareLength` are pointers to void. Since we use the function to compare two array elements of type `Vector2D` you must typecast them to pointers to vectors, e.g. "`(Vector2D *) v1`" before referencing these pointers.

```
int compareLength(const void* v1, const void* v2){
    double lengthV1=length(*(Vector2D *) v1);
    // NOTE: We first convert v1 to a pointer to Vector2D: "(Vector2D *) v1"
    // We then dereference this pointer using "*" in order to get the value
    // of the memory space to which it points to.
    // This is the same as:
    // Vector2D *vec1Ptr=(Vector2D *) v1;
    // Vector2D vec1=*vec1Ptr;
    // double lengthV1=length(vec1);
    double lengthV2=length(*(Vector2D *) v2);
    if (lengthV1<lengthV2) return -1;
    else if (lengthV1>lengthV2) return 1;
    else return 0;
}
```

(e) Write a function `printArray()` which outputs the array elements and their lengths and call the function in your main method.

```
void printArray(){
    printf("\nThe size of the array is: %d
            \nThe array elements are:",numVectors);
    for(int i=0;i<numVectors;i++)
          printf("\nvectors[%d]=(%lf, %lf) \tlength=%lf",i,
                  vectors[i].x, vectors[i].y, length(vectors[i]));
}
```

(f) Sort the array of vectors by length and output the sorted array.

Note: Please use the function `qsort` which implements a *quicksort* algorithm and is included by `<stdlib.h>`. The function can be used for sorting arrays of any type. This is achieved by giving it as arguments the array itself, the number of elements, the size of an array element in bytes, and a pointer to a function (given by the function name) which compares two elements of the respective type. The `qsort` function is very efficient and easy to use. You might want to read more details about it in the documentation :-)

```
qsort(vectors, numVectors, sizeof(Vector2D), compareLength);
printf("\n\nAfter sorting the array");
printArray();
```

Q2: Write a class `CFloatArray2D` which contains a two-dimensional array of float values of a user defined size (i.e. you have to allocate memory dynamically). This can be achieved by defining an array of pointers where each pointer points to a row of the 2D array.
Your class should have a default constructor which initialises the array to `NULL`, it should have a method `setSize(int nRows, int nColumns)` which sets the size of the array and allocates memory, it should have an output operator `<<` and it should have an operator

```
float& operator()(int row, int column)
```

which returns an element of the array. Write some code to test your class `CFloatArray2D`.

FloatArray2D.h

```
#pragma once

#include <iostream>
using namespace std;

class CFloatArray2D
{
public:
    CFloatArray2D();
    virtual ~CFloatArray2D();
    void setSize(int nRows, int nColumns);
    float& operator()(int row, int column);
    friend ostream& operator<<(ostream&, CFloatArray2D&);
private:
    int nRows,nColumns;
    float **data;
};
```

FloatArray2D.cpp

```
#include "FloatArray2D.h"
#include <stdio.h>

CFloatArray2D::CFloatArray2D(){ data=NULL; }

CFloatArray2D::~CFloatArray2D(void){
    for(int i=0;i<nRows;i++)
           delete[] data[i];
    delete[] data;
}

void CFloatArray2D::setSize(int nr, int nc){
    nRows=nr;
    nColumns=nc;
    data=(float**) malloc(nRows*sizeof(float*));
    for(int i=0;i<nRows;i++)
           data[i]=(float*) malloc(nColumns*sizeof(float));
}

float& CFloatArray2D::operator()(int row, int column){
    if ((row < 0) || (row >= nRows) || (column < 0) || (column >= nColumns))
           fprintf(stderr,"CFloatArray2D::(). out of range: size=(%d,%d),
                           index=(%d,%d)",nRows,nColumns,row,column);
    return data[row][column];
}

ostream& operator<<(ostream& s, CFloatArray2D& array)
{
    s << "\nnRows=" << array.nRows << " nColumns=" << array.nColumns;
    for(int i=0;i<array.nRows;i++){
           s << "\n";
           for(int j=0;j<array.nColumns;j++)
                  s << array(i,j) << "\t";
    }
    return s;
}
```